МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»  
(ФГБОУ ВО «ВГТУ», ВГТУ)

Факультет информационных технологий и компьютерной безопасности

Кафедра компьютерных интеллектуальных технологий проектирования

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №8

По дисциплине: «Системы хранения и обработки данных»

Тема: «Разработка физической структуры базы данных»

Выполнил работу студент группы змИИВТ-241: Фролов Д.С.

подпись, дата

Принял: Короленко В.В.

подпись, дата

Воронеж 2024

Лабораторная работа №8

«Разработка физической структуры базы данных»

1. Цель лабораторной работы: изучить и освоить процесс разработки физической структуры базы данных с использованием системы управления базами данных Postgres.

2. Задачи лабораторной работы:

- создание таблиц в СУБД Postgres с помощью SQL-запросов с атрибутами, связями, первичными и внешними ключами в соответствии с разработанной в предыдущей лабораторной работе логической структуры базы данных;

- построение графической диаграммы для отображения физической структуры базы данных.

3. Результаты лабораторной работы

Изучим основы работы с СУБД PostgreSQL. Сформируем SQL-запросы для создания таблиц в СУБД Postgres, соответствующих логической структуре базы данных, разработанной в предыдущей лабораторной работе. Результат создания таблиц представлен на рисунках 1-2.

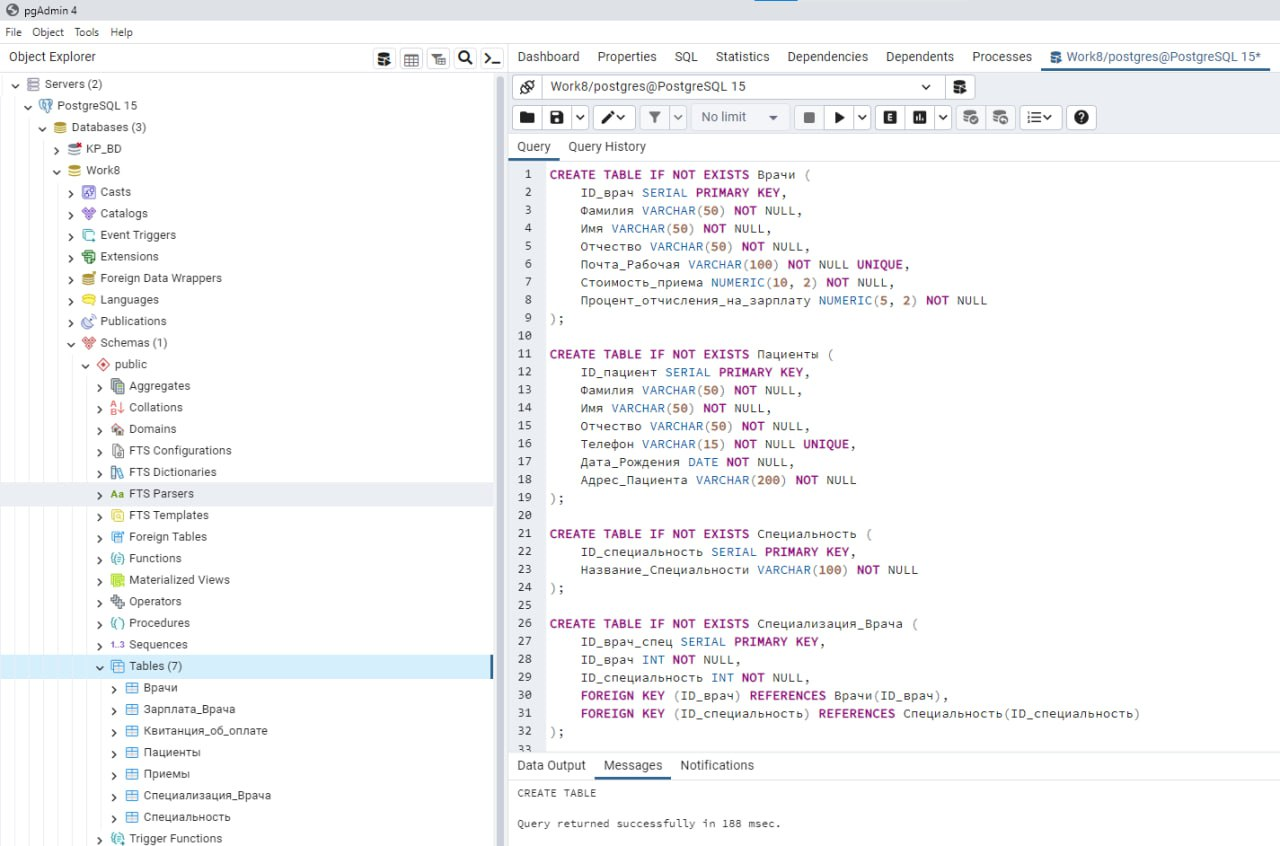


Рисунок 1 – Создание таблиц

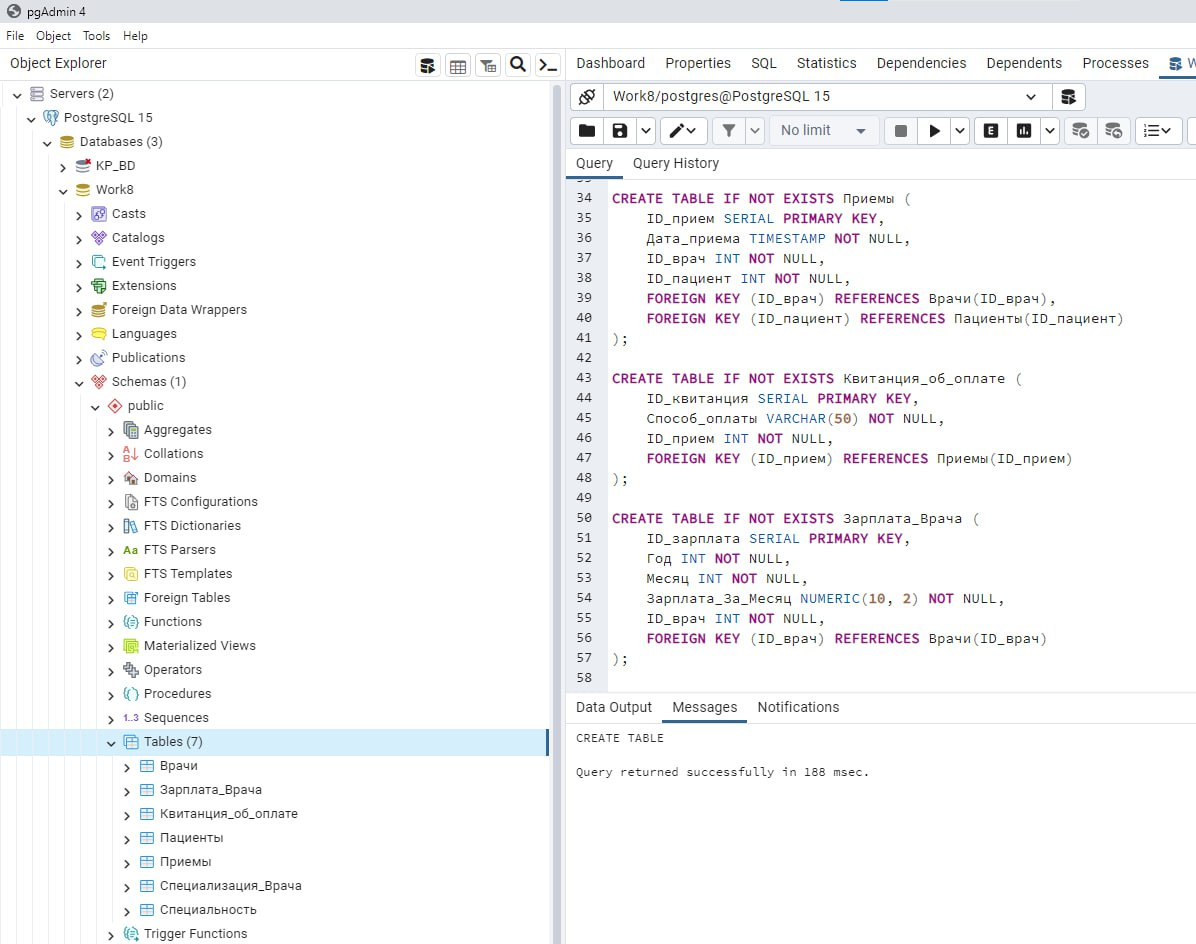


Рисунок 2 – Создание таблиц

Сохраним SQL-запросы в соответствующем файле со скриптами SQL, используя Visual Studio Code, в файле «work8.sql».

Опишем каждый запрос:

1. CREATE TABLE IF NOT EXISTS Врачи – создает таблицу «Врачи», если она не существует. Эта таблица хранит информацию о врачах.

* CREATE TABLE IF NOT EXISTS – создает новую таблицу только если она не существует;
* ID\_врач SERIAL PRIMARY KEY – создает столбец «ID\_врач» с уникальным автоматически увеличивающимся значением и делает его первичным ключом;
* Фамилия VARCHAR (50) NOT NULL – строковый столбец длиной до 50 символов, обязательный для заполнения;
* Имя VARCHAR (50) NOT NULL – строковый столбец длиной до 50 символов, обязательный для заполнения;
* Отчество VARCHAR (50) NOT NULL – строковый столбец длиной до 50 символов, обязательный для заполнения;
* Почта\_Рабочая VARCHAR (100) NOT NULL UNIQUE – строковый столбец длиной до 100 символов, обязательный для заполнения, должен быть уникальным;
* Стоимость\_приема NUMERIC (10,2) NOT NULL – число с точностью до двух знаков после запятой, обязательное для заполнения;
* Процент\_отчисления\_на\_зарплату NUMERIC (5,2) NOT NULL – число с точностью до двух знаков после запятой, обязательное для заполнения.

2. CREATE TABLE IF NOT EXISTS Пациенты – создает таблицу «Пациенты», если она не существует. Эта таблица хранит информацию о пациентах.

* ID\_пациент SERIAL PRIMARY KEY – создает столбец «ID\_пациент» с уникальным автоматически увеличивающимся значением и делает его первичным ключом;
* Фамилия VARCHAR (50) NOT NULL – строковый столбец длиной до 50 символов, обязательный для заполнения;
* Имя VARCHAR (50) NOT NULL – строковый столбец длиной до 50 символов, обязательный для заполнения;
* Отчество VARCHAR (50) NOT NULL – строковый столбец длиной до 50 символов, обязательный для заполнения;
* Телефон VARCHAR (15) NOT NULL UNIQUE – строковый столбец длиной до 15 символов, обязательный для заполнения, должен быть уникальным;
* Дата\_Рождения DATE NOT NULL – хранит дату рождения пациента, обязательное поле;
* Адрес\_Пациента VARCHAR (200) NOT NULL – строковый столбец длиной до 200 символов, обязательный для заполнения.

3. CREATE TABLE IF NOT EXISTS Специальность – создает таблицу «Специальность», если она не существует. Эта таблица содержит перечень возможных врачебных специальностей.

* ID\_специальность SERIAL PRIMARY KEY – создает столбец «ID\_специальность» с уникальным автоматически увеличивающимся значением и делает его первичным ключом;
* Название\_Специальности VARCHAR (100) NOT NULL – строковый столбец длиной до 100 символов, обязательный для заполнения.

4. CREATE TABLE IF NOT EXISTS Специализация\_Врача – создает таблицу «Специализация\_Врача», если она не существует. Данная таблица является промежуточной сущностью между таблицами «Врачи» и «Специальность».

* ID\_врач\_спец SERIAL PRIMARY KEY – создает столбец «ID\_врач\_спец» с уникальным автоматически увеличивающимся значением и делает его первичным ключом;
* ID\_врач INT NOT NULL – внешний ключ, ссылающийся на таблицу «Врачи» по столбцу «ID\_врач»;
* ID\_специальность INT NOT NULL – внешний ключ, ссылающийся на таблицу «Специальность» по столбцу «ID\_специальность»;
* FOREIGN KEY – ограничение, обеспечивающее ссылочную целостность.

5. CREATE TABLE IF NOT EXISTS Приемы – создает таблицу «Приемы», если она не существует. Данная таблица хранит информацию о приемах, включая дату и взаимосвязь врача с пациентом.

* ID\_прием SERIAL PRIMARY KEY – создает столбец «ID\_прием» с уникальным автоматически увеличивающимся значением и делает его первичным ключом;
* Дата\_приема TIMESTAMP NOT NULL – дата и время приема;
* ID\_врач INT NOT NULL – внешний ключ, ссылающийся на таблицу «Врачи» по столбцу «ID\_врач»;
* ID\_пациент INT NOT NULL – внешний ключ, ссылающийся на таблицу «Пациенты» по столбцу «ID\_пациент»;
* FOREIGN KEY – ограничение, обеспечивающее ссылочную целостность.

6. CREATE TABLE IF NOT EXISTS Квитанция\_об\_оплате – создает таблицу «Квитанция\_об\_оплате», если она не существует. Данная таблица хранит информацию о квитанциях за оплату приемов.

* ID\_квитанция SERIAL PRIMARY KEY – создает столбец «ID\_квитанция» с уникальным автоматически увеличивающимся значением и делает его первичным ключом;
* Способ\_оплаты VARCHAR (50) NOT NULL – строковый столбец длиной до 50 символов, обязательный для заполнения;
* ID\_прием INT NOT NULL – внешний ключ, ссылающийся на таблицу «Приемы» по столбцу «ID\_прием»;
* FOREIGN KEY – ограничение, обеспечивающее ссылочную целостность.

7. CREATE TABLE IF NOT EXISTS Зарплата\_Врача – создает таблицу «Зарплата\_Врача», если она не существует. Данная таблица хранит данные о зарплатах врачей за каждый месяц.

* ID\_зарплата SERIAL PRIMARY KEY – создает столбец «ID\_зарплата» с уникальным автоматически увеличивающимся значением и делает его первичным ключом;
* Год INT NOT NULL – год, за который выплачивается зарплата. Обязательное поле для заполнения;
* Месяц INT NOT NULL – месяц, за который выплачивается зарплата. Обязательное поле для заполнения;
* Зарплата\_За\_Месяц NUMERIC (10, 2) NOT NULL – числовое значение зарплаты с двумя знаками после запятой. Обязательное поле для заполнения;
* ID\_врач INT NOT NULL – внешний ключ, ссылающийся на таблицу «Врачи» по столбцу «ID\_врач»;
* FOREIGN KEY – ограничение, обеспечивающее ссылочную целостность.

Обернем созданную базу данных в docker-контейнер. Для этого необходимо выполнить команду docker-compose up -d. Результат представлен на рисунках 3-4.

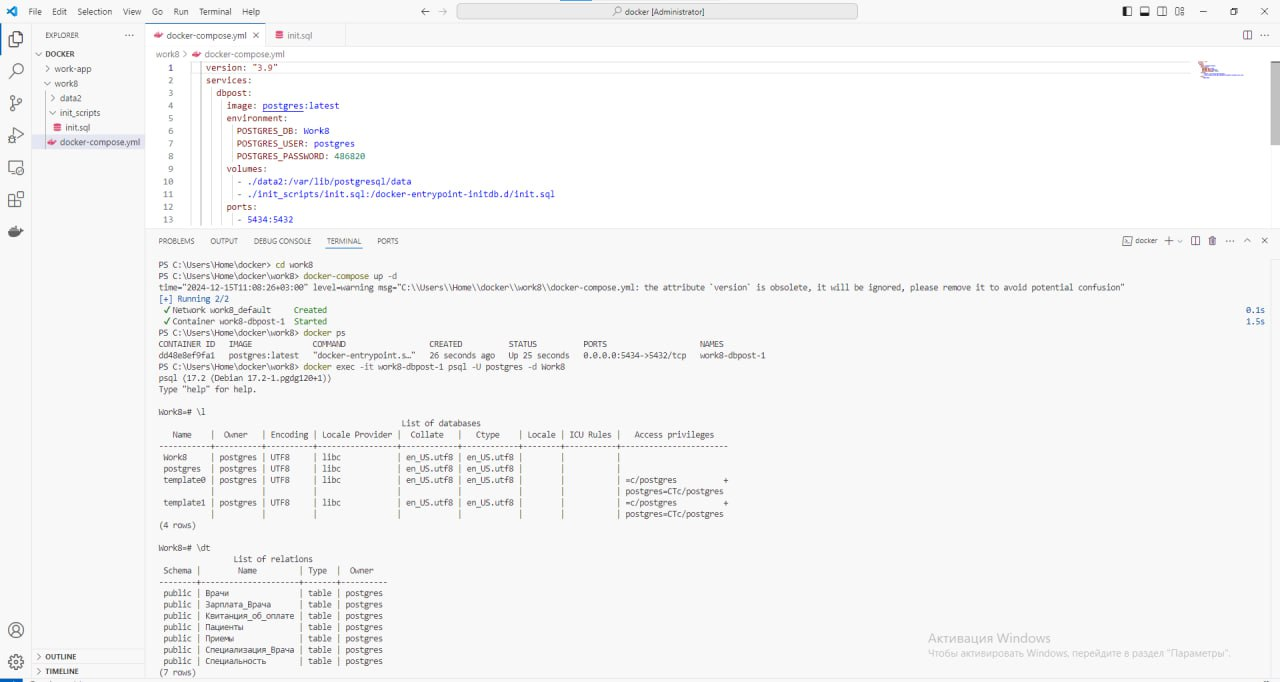


Рисунок 3 – Оборачивание базы данных в docker-контейнер

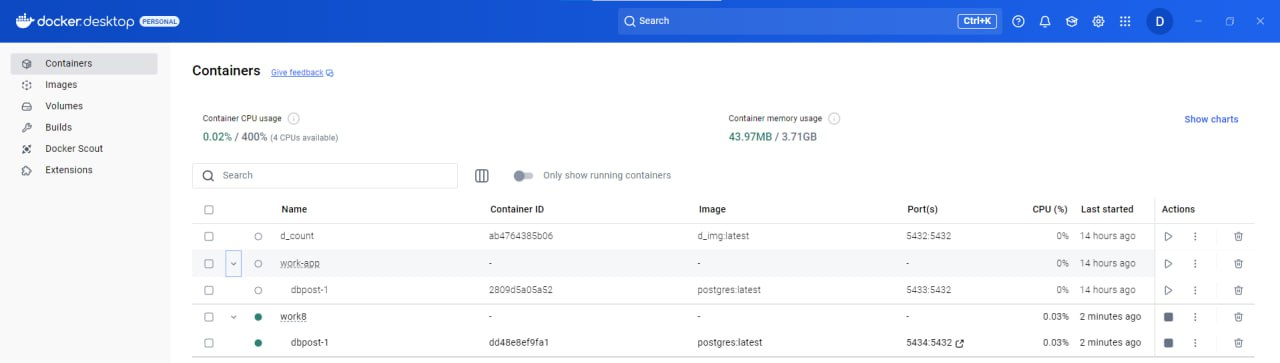


Рисунок 4 – Результат запуска Docker Compose

Проверим через DBeaver развернутую СУБД. Для этого запустим DBeaver и создадим новое подключение. В списке доступных баз данных выберем PostgreSQL (рисунок 5).

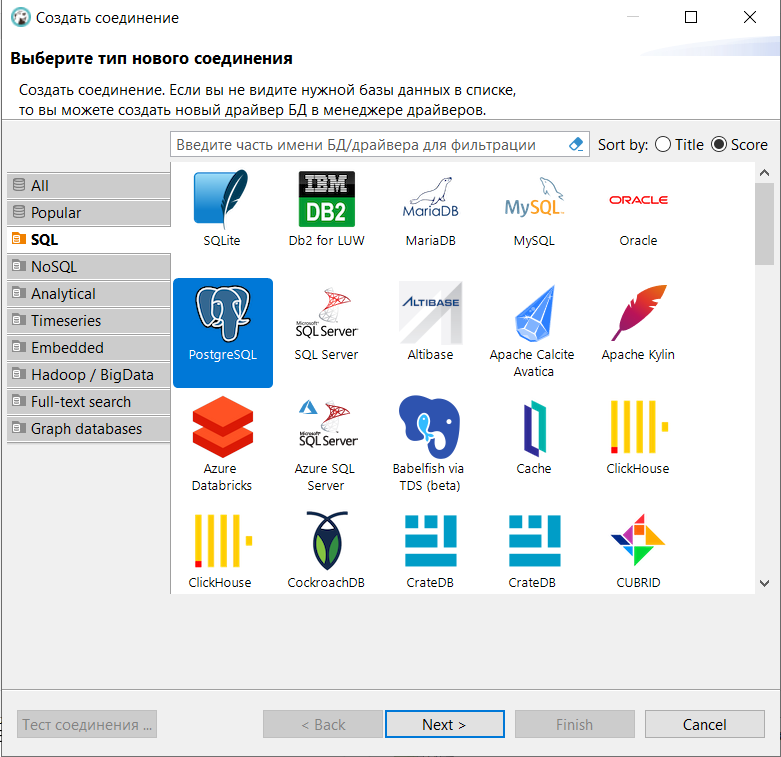


Рисунок 5 – Выбор PostgreSQL в качестве базы данных

Далее нажимаем «Next», выполняем настройки соединения и нажимаем на кнопку «Тест соединения» (рисунок 6).

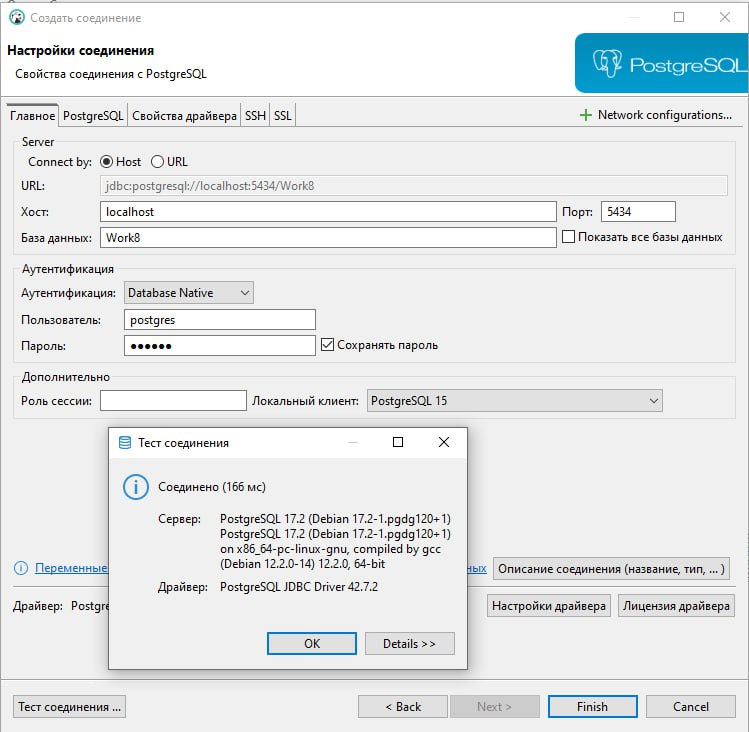


Рисунок 6 – Настройки и тест соединения

Результат проверки развернутой СУБД через DBeaver представлен на рисунке 7.

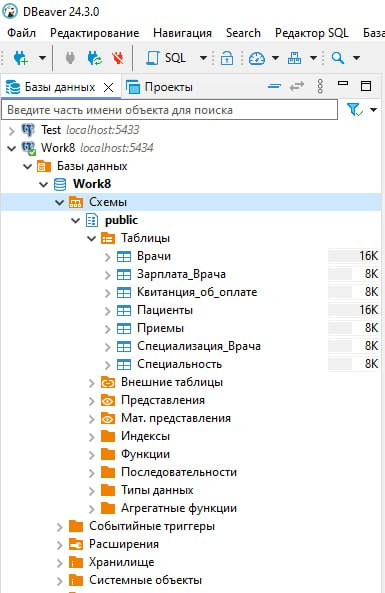


Рисунок 7 – Результат проверки развернутой СУБД через DBeaver

Построим графическую диаграмму, отображающую физическую структуру базы данных (рисунок 8).

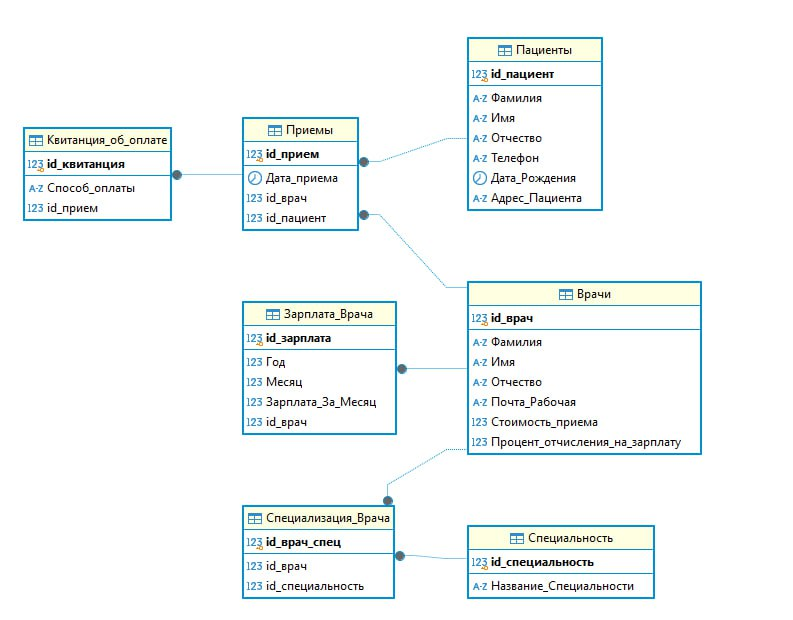


Рисунок 8 – Физическая структура базы данных «Платный прием в поликлинике»

Диаграмма отражает физическую структуру базы данных «Платный прием в поликлинике» и включает в себя следующие основные сущности: «Врачи», «Пациенты», «Приемы», «Специальность», «Квитанция\_об\_оплате», «Зарплата\_Врача» и дополнительную сущность «Специализация\_Врача» для реализации связи между врачами и их специальностями.

Сущность «Врачи» содержит данные о врачах, включая стоимость приема и процент отчисления на зарплату, что помогает рассчитывать их доход. Атрибут «id\_врач» является первичным ключом, который уникально идентифицирует врача. Атрибуты «Фамилия», «Имя» и «Отчество» являются личными данными врача для идентификации. Атрибут «Почта\_Рабочая» предназначен для связи внутри учреждения и с пациентами в рамках профессиональной деятельности. Атрибут «Стоимость\_приема» предназначен для хранения информации о стоимости приема, которая устанавливается персонально для каждого врача. Атрибут «Процент\_отчисления\_на\_зарплату» используется для расчета заработной платы.

Сущность «Пациенты» хранит данные о пациентах, включая контактные данные и дату рождения. Атрибут «id\_пациент» является первичным ключом, который уникально идентифицирует пациента. Атрибуты «Фамилия», «Имя» и «Отчество» являются личными данными пациента для идентификации. Атрибут «Телефон» предназначен для связи и оповещений, связанных с приемами. Атрибут «Дата\_Рождения» позволяет учитывать возрастные особенности при лечении. Атрибут «Адрес\_Пациента» предназначен для возможной отправки документов по почте.

Сущность «Приемы» хранит данные о визитах: кто, кто и в какую дату был на приеме. Атрибут «id\_прием» является первичным ключом, который уникально идентифицирует прием. Атрибут «Дата\_приема» хранит информацию о дате и времени посещения пациента. Атрибуты «id\_врач» и «id\_пациент» являются внешними ключами. Данные внешние ключи связывают прием с конкретным врачом и пациентом.

Сущность «Специальность» позволяет учитывать медицинские специализации врачей. Атрибут «id\_специальность» является первичным ключом, который уникально идентифицирует специальность. Атрибут «Название\_Специальности» хранит информацию о наименовании медицинской специализации.

Сущность «Квитанция\_об\_оплате» предназначена для учета оплаты за прием и привязана к конкретному визиту пациента. Атрибут «id\_квитанция» является первичным ключом, который уникально идентифицирует квитанцию. Атрибут «Способ\_оплаты» хранит информацию о способе оплаты за прием (наличные, карта и т.п.). Атрибут «id\_прием» является внешним ключом. Данный внешний ключ связывает квитанцию с конкретным приемом.

Сущность «Зарплата\_Врача» хранит информацию о заработной плате врача по месяцам. Атрибут «id\_зарплата» является первичным ключом, который уникально идентифицирует записи о зарплате. Атрибуты «Год» и «Месяц» предназначены для хранения информации о периоде, за который начислена зарплата. Атрибут «id\_врач» является внешним ключом, так как запись о зарплате привязана к конкретному врачу.

Сущность «Специализация\_Врача» является промежуточной сущностью для реализации связи «Многие ко многим». Атрибут «id\_врач\_спец» является первичным ключом, который уникально идентифицирует запись. Данная сущность содержит внешние ключи «id\_врач» (ссылка на сущность «Врачи») и «id\_cпециальность» (ссылка на сущность «Специальность)».

Сущность «Врачи» связана с сущностью «Специализация\_Врача» связью «Один ко многим» – один врач может иметь несколько специализаций. Сущность «Специальность» связана с сущностью «Специализация\_Врача» связью «Один ко многим» – одна специальность может быть у нескольких врачей. Введение сущности «Специализация\_Врача» позволяет устранить связь «Многие ко многим» между врачами и специальностями, нормализуя базу данных.

Сущность «Врачи» связана с сущностью «Приемы» связью «Один ко многим» – один врач может проводить несколько приемов.

Сущность «Пациенты» связана с сущностью «Приемы» связью «Один ко многим» – один пациент может посетить множество приемов.

Сущность «Приемы» связана с сущностью «Квитанция\_об\_оплате» связью «Один к одному» – каждому приему соответствует одна квитанция об оплате.

Сущность «Врачи» связана с сущностью «Зарплата\_Врача» связью «Один ко многим» – один врач может иметь множество записей о зарплате за разные месяцы.

Первичные ключи обеспечивают уникальность записей в каждой таблице:

* «id\_врач» – первичный ключ для сущности «Врачи»;
* «id\_пациент» – первичный ключ для сущности «Пациенты»;
* «id\_прием» – первичный ключ для сущности «Приемы»;
* «id\_специальность» – первичный ключ для сущности «Специальность»;
* «id\_квитанция» – первичный ключ для сущности «Квитанция\_об\_оплате»;
* «id\_зарплата» – первичный ключ для сущности «Зарплата\_Врача»;
* «id\_врач\_спец» – первичный ключ для сущности «Специализация\_Врача».

Внешние ключи устанавливают отношения между таблицами, позволяя выполнять запросы, объединять данные и поддерживать ссылочную целостность:

* В сущности «Приемы» – «id\_врач» (ссылка на таблицу «Врачи») и «id\_пациент» (ссылка на таблицу «Пациенты»);
* В сущности «Квитанция\_об\_оплате» – «id\_прием» (ссылка на таблицу «Приемы»);
* В сущности «Зарплата\_Врача» – «id\_врач» (ссылка на таблицу «Врачи»);
* В сущности «Специализация\_Врача» – «id\_врач» (ссылка на таблицу «Врачи») и «id\_специальность» – ссылка на таблицу «Специальности».

Присвоение ключей соответствует нормализации базы данных, избегая избыточности и аномалий при обновлении данных. Избавление от связи «Многие ко многим» между «Врачи» и «Специальность» посредством введения сущности «Специализация\_Врача» соответствует принципам реляционных баз данных. Это позволяет хранить информацию о том, какие врачи обладают какими специальностями, без дублирования данных.

4. Выводы по результатам лабораторной работы

В ходе выполнения данной лабораторной работы был изучен и освоен процесс разработки физической структуры базы данных с использованием системы управления базами данных Postgres.

5. Ответы на контрольные вопросы

1. PostgreSQL – это объектно-реляционная система управления базами данных с открытым исходным кодом. Особенности PostgreSQL:

- многоверсионное управление параллелизмом (MVCC). Применяется для снятия блокировок данных при транзакциях в конкурентных (многопользовательских) условиях. MVCC изолирует транзакции для каждой версии БД и защищает их от несогласованного изменения данных, то есть блокировка на чтение не блокирует запись, и наоборот;

- высокий уровень производительности, обеспеченный использованием индексов (стандартных, частичных и функциональных), интеллектуальным планированием запросов, системами управления буферами памяти и кэширования. База данных PostgreSQL отлично подходит для любых рабочих нагрузок: OLTP, OLAP, хранилища данных и многое другое. Она также полностью соответствует стандартам SQL, что позволяет писать эффективные запросы и программы PL/PGSQL. А переиндексация и реорганизация таблиц может быть выполнена в режиме онлайн;

- в PostgreSQL предусмотрены расширенные настройки для создания собственных плагинов и персонализации параметров, а также возможность индексирования и поддержки географических, растровых и векторных объектов с помощью расширения PostGIS. Эта СУБД совместима с ОС Windows, MacOS, Solaris и Linux;

- работа с большими объемами. В PostgreSQL ограничений по объему базы и количеству записей в ней нет;

- поддержка сложных запросов. PostgreSQL поддерживает рекурсивные запросы, оконные функции, общие табличные выражения (CTE) и другие продвинутые возможности SQL, что позволяет выполнять сложные сложные аналитические задачи;

- написание функций на нескольких языках. Кроме стандартного SQL, в PostgreSQL можно писать на C, C++, Java, Python, PHP, Lua и Ruby;

- возможность одновременного доступа к базе с нескольких устройств. В СУБД реализована клиент-серверная архитектура, когда база данных хранится на сервере, а доступ к ней осуществляется с клиентских компьютеров;

- транзакционная целостность. Полная поддержка ACID-транзакций обеспечивает надежность и согласованность данных;

- репликация и высокая доступность. Поддержка как синхронной, так и асинхронной репликации позволяет настраивать системы с высокой доступностью и масштабируемостью;

- безопасность. Расширенные возможности аутентификации, управления ролями и шифрования данных обеспечивают высокий уровень безопасности.

2. Data Definition Language (DDL) – это группа операторов определения данных. Другими словами, с помощью операторов, входящих в эту группы, мы определяем структуру базы данных и работаем с объектами этой базы, т.е. создаем, изменяем и удаляем их. В эту группу входят следующие операторы:

- CREATE – используется для создания объектов базы данных;

- ALTER – используется для изменения объектов базы данных;

- DROP – используется для удаления объектов базы данных.

3. Data Manipulation Language (DML) – это группа операторов для манипуляции данными. С помощью этих операторов мы можем добавлять, изменять, удалять и выгружать данные из базе, т.е. манипулировать ими. В эту группу входят самые распространенные операторы языка SQL:

- SELECT – осуществляет выборку данных;

- INSERT – добавляет новые данные;

- UPDATE – изменяет существующие данные;

- DELETE – удаляет данные.

4. Data Control Language (DCL) – группа операторов определения доступа к данным. Иными словами, это операторы для управления разрешениями, с помощью их мы можем разрешать или запрещать выполнение определенных операций над объектами базы данных. Сюда входят:

- GRANT – предоставляет пользователю или группе разрешения на определенные операции с объектом;

- REVOKE – отзывает выданные разрешения;

- DENY – задает запрет, имеющий приоритет над разрешением.

5. Как правило, SQL-запросы начинаются с команды SELECT. Она указывает, какие столбцы из таблицы должны быть возвращены в наборе результатов.

6. Для создания базы данных в SQL используется команда «CREATE DATABASE». Эта команда создает новую базу данных с указанным именем.

7. Для создания таблицы в базе данных используется команда CREATE TABLE. Эта команда позволяет определить структуру таблицы, включая ее столбцы, типы данных, ограничения и другие параметры.

8. Функция mod возвращает остаток от деления n на m. Оператор IN предназначен для проверки значения на совпадение с заранее заданным списком значений. Таким образом, правильные варианты ответов:

- SELECT \* FROM employee WHERE mod(mod(sal, 23), 5) = 0;

- SELECT \* FROM employee WHERE sal IN (1500, 3000).

9. Параметр ASC сортирует результирующий набор в порядке возрастания по expression (по умолчанию, если модификатор не указан). Если указан параметр NULLS FIRST, то все значения NULL сортируются перед значениями, отличными от NULL, в результирующем наборе. Если указан параметр NULLS LAST, то все значения NULL сортируются после значений, отличных от NULL, в результирующем наборе.

Таким образом, правильный вариант ответа – SELECT \* FROM employee ORDER BY dept\_id NULLS FIRST.

10. Так как длина типа данных char не задана, то подразумевается, что она равна единице, то есть char(1), тогда верны следующие варианты ответов:

- '1';

- 'a'.

11. Следующие конструкции запроса не являются верными:

- Select список полей from список таблиц having условия группировки group by поля для группировки.

Здесь использовано ключевое слово having перед group by, что нарушает правильный порядок. Ключевое слово having должно следовать после group by;

- Select список полей from список таблиц order by поля для сортировки union Select список полей from список таблиц order by поля для сортировки.

В SQL нельзя использовать order by внутри каждого запроса при использовании union. Общий order by должен следовать после всех объединений.

12. Select avg (advance), sum (sales) from titles group by type.